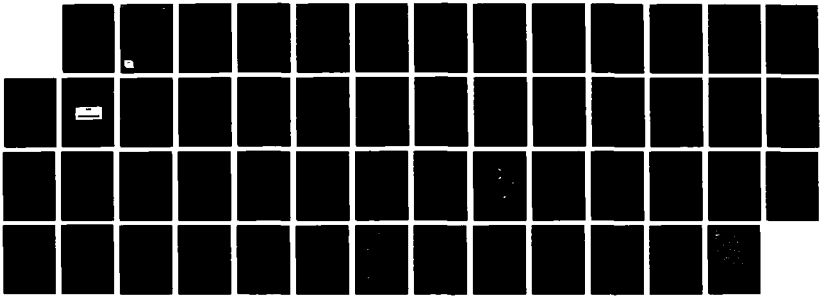
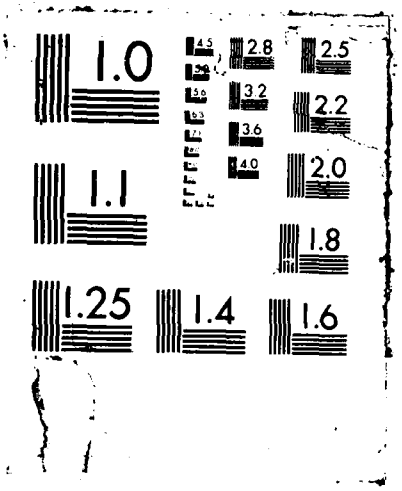
AD-A189 350 CAIS (COMMON APSE (ADA (TRADE NAME) PROGRAMMING SUPPORT ENVIRONMENT) INTE..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA J F KRAMER ET AL. 14 AUG 87 IDA-P-2034 1/1

UNCLASSIFIED IDA/HQ-87-32618 MDA903-84-C-0031 F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0

UNCLASSIFIED

IDA PAPER P-2034

CAIS READER'S GUIDE FOR DoD-STD-1838

John F. Kramer
Patricia Oberndorf
John Long
Clyde Roby
R. Max Robinson
Jeff Clouse
John Chludzinski

August 1987

Prepared for
Ada* Joint Program Office

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

*Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

Series B
IDA Log No. HQ 87-32618

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, or (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Papers

Papers normally address relatively restricted technical or policy issues. They communicate the results of special analyses, interim reports or phases of a task, ad hoc or quick reaction work. Papers are reviewed to ensure that they meet standards similar to those expected of refereed papers in professional journals.

Memorandum Reports

IDA Memorandum Reports are used for the convenience of the sponsors or the analysts to record substantive work done in quick reaction studies and major interactive technical support activities; to make available preliminary and tentative results of analyses or of working group and panel activities; to forward information that is essentially unanalyzed and unevaluated; or to make a record of conferences, meetings, or briefings, or of data developed in the course of an investigation. Review of Memorandum Reports is suited to their content and intended use.

The results of IDA work are also conveyed by briefings and informal memoranda to sponsors and others designated by the sponsors, when appropriate.

The work reported in this document was conducted under contract MDA 903 84 C 0031 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

This paper has been reviewed by IDA to assure that it meets high standards of thoroughness, objectivity, and sound analytical methodology and that the conclusions stem from the methodology.

Public release/unlimited distribution; unclassified.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

AD-A187350

1a REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Public release/unlimited disitribution
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) P-2034	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Institute for Defense Analyses 6b OFFICE SYMBOL IDA	7a NAME OF MONITORING ORGANIZATION
6c ADDRESS (City, State, and Zip Code) 1801 N. Beauregard St. Alexandria, VA 22311	7b ADDRESS (City, State, and Zip Code)
8a NAME OF FUNDING/SPONSORING ORGANIZATION Ada Joint Program Office 8b OFFICE SYMBOL (if applicable) (AJPO)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER MDA 903 84 C 0031
8c ADDRESS (City, State, and Zip Code) 1211 Fern St., Room C-107 Arlington, VA 22202	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. T-D5-305 / WORK UNIT ACCESSION NO.
11 TITLE (Include Security Classification) CAIS Reader's Guide for DoD-STD-1838 (U)	
12 PERSONAL AUTHOR(S) J. Kramer, P. Oberndorf, J. Long, C. Roby, R. Robinson, J. Clouse, J. Chludzinski	
13a TYPE OF REPORT Final 13b TIME COVERED FROM_____ TO _____	14 DATE OF REPORT (Year, Month, Day) 1987 August 14 15 PAGE COUNT 54
16 SUPPLEMENTARY NOTATION	

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Ada Programming Language; Common Ada Programming Support Environment (APSE) Interface Set (CAIS); DoD-STD-1838; Software Environments; Software Tools; Software Development; Kernel APSE (Kapse); Minimal APSE (MAPSE).

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The CAIS Reader's Guide for DoD-STD-1838 has been produced to aid readers in understanding DoD-STD-1838, the Military Standard Common Ada Programming Support Environment (APSE) Interface Set (CAIS). DoD-STD-1838 defines a set of interfaces which allows software tools resident in an APSE access to common operating system services and facilities. APSE tools are Ada programs, each of which is used for a specific software development task. These tools need facilities to communicate with their environment, including other tools in that environment, which the Ada language does not provide. The CAIS can thus be regarded as providing extended interfaces between an external environment and a host system.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ■ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Dr. John F. Kramer, IDA	22b TELEPHONE (Include Area Code) (703) 824-5504 22c OFFICE SYMBOL

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

IDA PAPER P-2034

CAIS READER'S GUIDE FOR DoD-STD-1838

John F. Kramer
Patricia Oberndorf John Long
Clyde Roby R. Max Robinson
Jeff Clouse John Chludzinski

August 1987

IDA
INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-D5-305

Preamble

The *CAIS Reader's Guide for DOD-STD-1838* has been produced to aid readers in understanding DOD-STD-1838, the *Military Standard Common Ada[1] Programming Support Environment (APSE) Interface Set (CAIS)*[2]. It is neither the intent nor purpose of this guide to provide a comprehensive presentation of the CAIS. Instead, the purpose is to give an overview of the model defined in the CAIS document. DOD-STD-1838 contains the sections Scope, Referenced Documents, Definitions, General Requirements, Detailed Requirements and Notes. The CAIS node model and security model are discussed in the General Requirements section. Within the Detailed Requirements section, various topics are described, including node management, processes, and input and output. Chapter 1 of this guide provides an introduction to the CAIS. The CAIS node model and security model are discussed in Chapter 2. In Chapter 3, General Node Management, CAIS node management operations are discussed. Chapter 4 provides further explanation of CAIS processes. Finally, Input and Output provided by the CAIS are discussed in Chapter 5. Having read the *CAIS Reader's Guide for DOD-STD-1838*, one should have a sufficient understanding of the CAIS to begin reading the DOD-STD-1838 CAIS document.

---

[1]Ada is a registered trademark of the United States Government, Ada Joint Program Office.

[2]*Military Standard Common Ada Programming Support Environment (APSE) Interface Set (CAIS)*, United States Department of Defense, DOD-STD-1838, 9 October 1986.

Contents

Figures

Tables

1. INTRODUCTION

1.1 History of the CAIS

The High Order Language Working Group (HOLWG), formed by the Department of Defense (DoD), recognized early in the Ada program that, while the Ada language itself did not require a special development and maintenance environment, the life-cycle maintenance of mission critical computer systems did.

> It was felt that an integrated software environment containing a set of good tools would encourage acceptance of the language, thereby magnifying the benefits of the language standardization effort. [SWAV]

Central to the development environment for the Ada language is the concept of a Programming Support Environment. A Programming Support Environment involves a collection of software tools that aid a developer or life cycle maintainer in his programming tasks during coding, testing and debugging. These tools traditionally include compilers, editors, debuggers, configuration control aids, linkers and text formatters.

An Ada Programming Support Environment (APSE) is an environment for the development of mission critical software systems written in the Ada language. Because of the volume of Ada software that the DoD intends to procure, APSEs must be available at several installations and on many different types of hardware.

> The purpose of an APSE is to support the development and maintenance of Ada applications software throughout its life cycle, with particular emphasis on software for embedded computer applications. [STONEMAN]

The requirements document for an APSE, "STONEMAN" [STONEMAN], goes further than being just a requirements document, for it also specifies an architecture for an APSE. In this architecture, a layering is defined which is generally portrayed as two layers surrounding a central core as shown in Figure 1.

The core is referred to as a Kernel APSE (KAPSE). The KAPSE represents operating system services commonly available to tools and applications programs. Thus, the KAPSE provides a common set of capabilities regardless of what the host may be. Such capabilities include general operating system services such as file management services and process and device control services. While each host is expected to have different implementations and similar or identical capabilities, the interfaces and their functionality provided in a KAPSE, according to [STONEMAN], must be identical from one host to another. The Common APSE Interface Set (CAIS) [1838] addresses the concerns expressed in [STONEMAN] for a common set of interfaces at this level (see Figure 1).

The next layer, the Minimal APSE (MAPSE), consists of software tools which minimally support software development, such as compilers, editors and linkers. According to [STONEMAN], these tools are to be written in the Ada language and are to be transportable, at the source level, to other APSEs. To achieve transportability, MAPSE tools are required to use the common interfaces in the KAPSE.

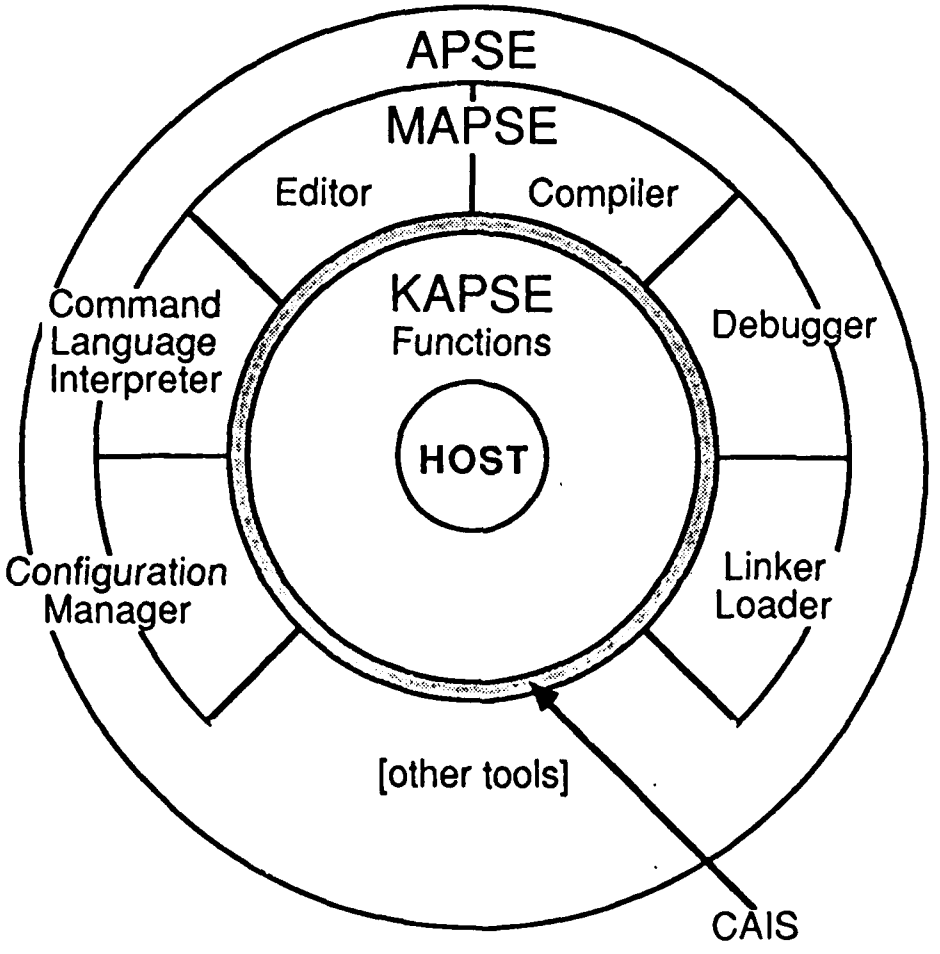


FIGURE 1. APSE Model

The top layer, the APSE, is to provide project unique tools and services. As such, the APSE may be viewed as a comprehensive set of tools for integrated life cycle development of software.

1.2 The Need for a CAIS

When DoD started procuring tools for the Ada program, it did not restrict itself to procuring individual tools. Rather, the DoD embarked upon the procurement of APSEs. Two procurements were started: one by the Army, called the Ada Language System (ALS), and the other by the Air Force, called the Ada Integrated Environment (AIE). Unfortunately, the interfaces provided from the KAPSE to the MAPSE were different in these two APSEs. Because of divergent approaches at the KAPSE interface level by the ALS and AIE contractors, a team was formed (by tri-services agreement [APSE]) to define more specific KAPSE interface requirements. This team is the KAPSE Interface Team (KIT) and is chaired by the Naval Ocean Systems Center (NOSC), a Navy laboratory. Added to the KIT was the KAPSE Interface Team from Industry and Academia (KITIA). The KIT/KITIA periodically issues reports on their progress, but their most significant product appeared in June 1987 with the public presentation of DOD-STD-1838, the Military Standard Common Ada Programming Support Environment (APSE) Interface Set (CAIS) [1838].

The KIT/KITIA had examined the differences in the ALS KAPSE and the AIE KAPSE and concluded that a mere host encapsulation would be insufficient to achieve the objectives of portability and interoperability. In addition, a common process composition and data interrelation model was needed. The two KAPSEs were sufficiently different that a common KAPSE subset definition would not achieve the objectives at all. Furthermore, the common interfaces of the two KAPSEs could not completely encapsulate the host. Some of the needed functionality is inherently host-dependent and therefore is very difficult to provide in a common form across all conceivable hosts. Thus, the KIT/KITIA developed the CAIS based on these principles but without being constrained to map directly onto the KAPSE designs of either the ALS or the AIE.

One interpretation of the CAIS in relation to the KAPSE interface layer is that the CAIS provides an interface layer above the level of the KAPSE due to the addition of the data interrelation model which traditionally has been a part of individual tools rather than of the common underpinnings of these tools. It thereby replaces large portions of the KAPSE from the tool-writer's point of view. More primitive KAPSE-like interfaces could still be valuable to enhance portability of the CAIS implementations themselves or of tools that access host-dependent facilities.

An extensive review of the proposed Military Standard CAIS [CAIS85] was made by the government, DoD, industry and industry groups, and academia as well as individuals. During the formal military standardization process, nearly 600 comments were submitted against this document. As a result, the proposed Military Standard CAIS was voted unanimously by the CAIS Standardization Working Group to become a military standard on 9 October 1986. DOD-STD-1838 [1838] was delivered to the Ada Joint Program Office on 16 June 1987.

The CAIS document [1838] defines a set of interfaces which allows software tools resident in an APSE access to common operating system services and facilities. APSE tools are Ada programs, each of which is used for a specific software development task. These tools need

facilities to communicate with their environment, including other tools in that environment, which the Ada language does not provide. The CAIS can thus be regarded as providing extended interfaces between an external environment and a host system. This intent of the CAIS is illustrated in Figure 2.

The interfaces in [1838] are in the form of a set of Ada package specifications. The capabilities specified are those which are commonly encountered in operating systems. These include operating systems such as MS-DOS[3], UNIX[4], VMS[5] and VM/CMS[6], which are in widespread use in industry and which are used as software development environments. Thus, the CAIS includes facilities for process initiation and control and file and device management.

The CAIS is not intended to provide an exhaustive set of all operating system facilities available today. Rather it is the goal of the CAIS to provide those facilities which have been found most useful in most operating systems and which have an impact on moving tool sets or project databases between APSEs.

The CAIS must be seen as the common portion of a set of host-encapsulating interfaces. There is good reason, and in some cases even a necessity, for a CAIS implementation to provide interfaces that are not part of the CAIS, but nevertheless integrate with the CAIS model. Whether or not these additional interfaces extend the CAIS to be the equivalent of a complete operating system depends upon the implementation. An implementation piggy-backed onto or integrated with an existing host operating system is unlikely to replicate many host-dependent interfaces that do not interact with the CAIS model and are available by calling the host operating system directly.

The objective of the CAIS is to provide a common set of interfaces in APSEs and thus promote the transportability of tools and the ability to move development project databases between systems that support the CAIS. Figure 3 depicts such a common tool with a common set of interfaces for two different tailored CAIS implementations. A common set of interfaces is necessary because APSEs are expected to be hosted on a variety of machines and operating systems. Large systems are often developed by several organizations and often maintained during the remainder of their life cycle by someone other than the developer. Because tools in APSEs regularly require facilities of the host system (e.g., the directory structure or the naming conventions of the host's file management system), those tools and the project database generated by them become host dependent. Since such facilities are not within the scope of the Ada language, the absence of a standard such as the CAIS will make it difficult to achieve true transportability and to move the project database.

---

[3]*MS-DOS is a registered trademark of Microsoft, Inc.*

[4]UNIX is a registered trademark of AT&T.

[5]VMS is a registered trademark of Digital Equipment Corporation.

[6]VM/CMS is a registered trademark of International Business Machines, Inc.

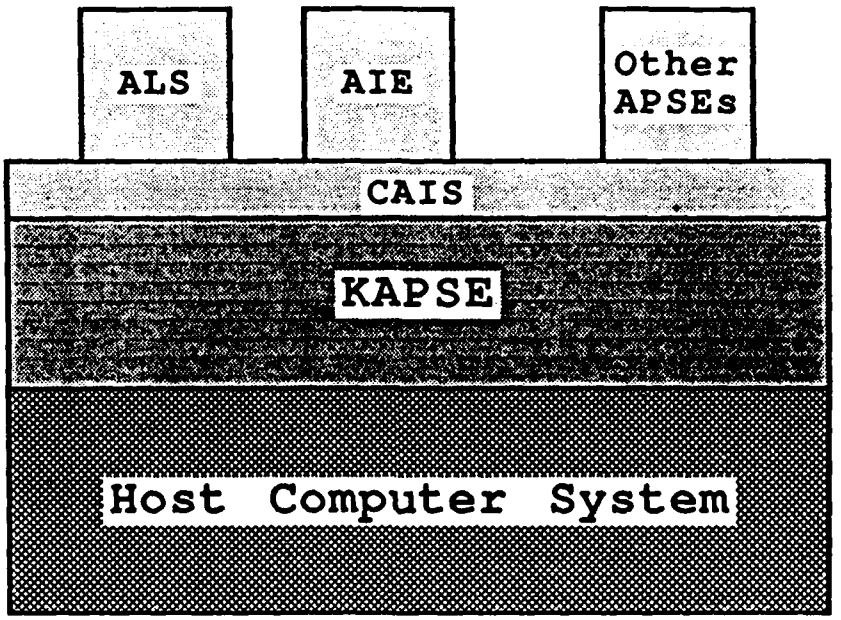


FIGURE 2. Purpose of CAIS

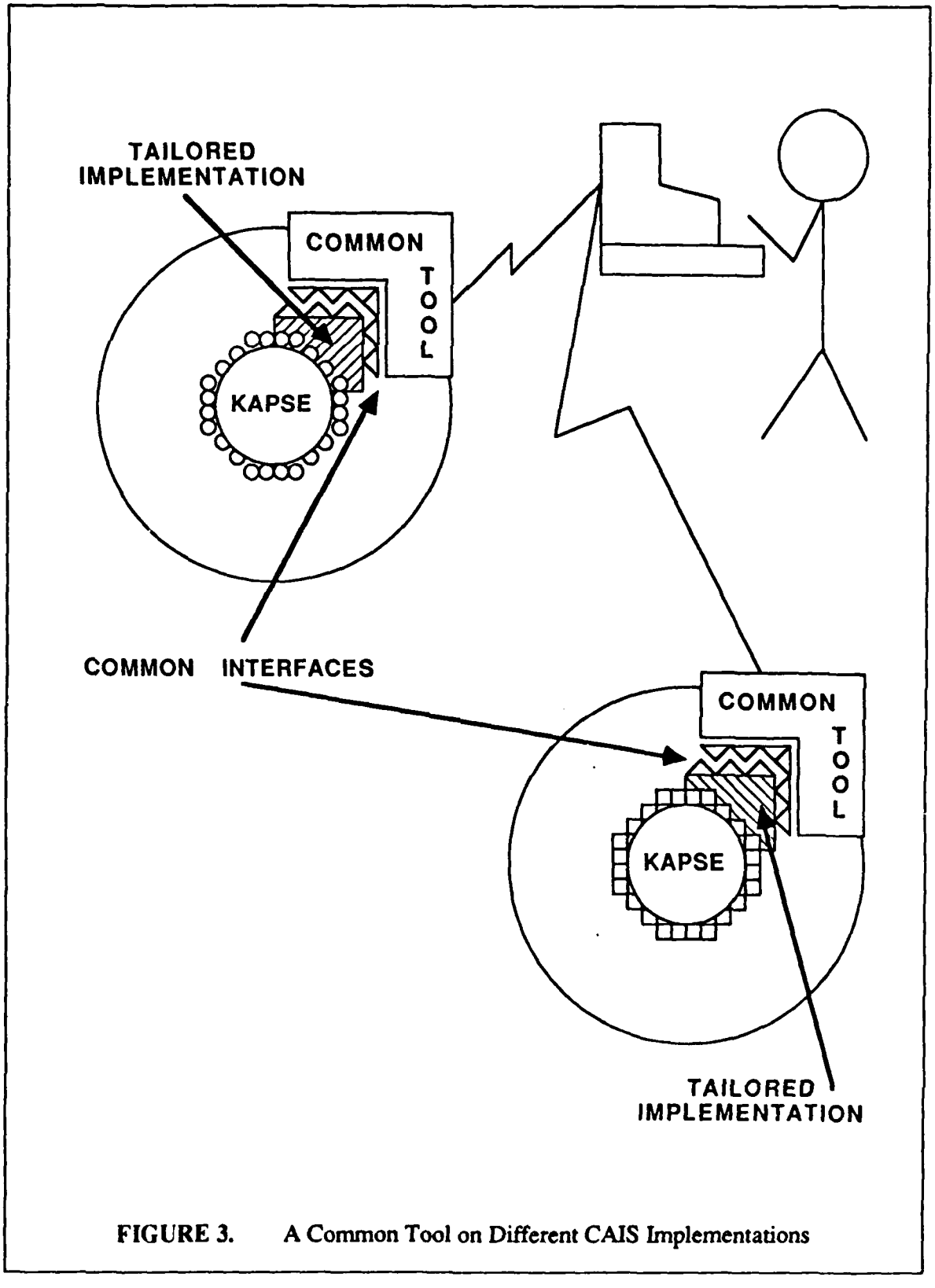


FIGURE 3. A Common Tool on Different CAIS Implementations

1.3 Future of the CAIS

It is desirable that the CAIS be used in all DoD-sanctioned APSEs. Towards this purpose, the CAIS document has been submitted for and has become a military standard, DOD-STD-1838.

At the same time, the KIT/KITIA has also developed a requirements document [RAC86] for the evolution of the CAIS. While the contents of the requirements document reflect closely the operations provided in DOD-STD-1838, differences exist between the two documents. A contractor has been chosen by a competitive process to further refine and develop the CAIS. The objective is to arrive at a DOD-STD-1838A by early 1989.

Since the CAIS has been adopted as a military standard, a common approach to the foundations of APSEs exists, thus encouraging and promoting transportability of software tools and project databases. Another major benefit is the ability to establish a larger tool marketplace because of the larger number of APSEs in which a tool can be installed. It is expected that software development productivity using the CAIS-based APSEs will also increase due to the ability to share good tools more widely, thus further enhancing the software productivity goal addressed by the creation of the Ada language.

There remain many problems of interoperability and transportability which the evolution of the CAIS must address or for which other standards might be proposed. In particular is the desire to provide inter-tool data standards so that tool builders could insert an individual component of a tool set into someone else's tool set. DOD-STD-1838 is a good starting basis for such improvements because it has been widely reviewed and includes numerous improvements over other environment interfaces such as UNIX while not being too advanced to prevent a reasonable number of implementations for wide use.

Several implementations of the CAIS have been and continue to be developed. Gould, Inc. finished the most complete implementation of [CAIS85] in early 1986 and is in the process of updating this implementation to [1838]; this should be completed in late 1987. TRW's implementation of [1838] will be complete in 1988; TRW already has completed a partial prototype implementation of [CAIS85]. Dr. Timothy Lindquist, currently at Arizona State University, has developed an Operational Definition of [CAIS85] and is currently in the process of updating it for [1838]. Although the MITRE Corporation has not developed a complete implementation of [CAIS85], they have been instrumental in developing partial prototype implementations which address feasibility issues. The first partial prototype implementation by MITRE addressed the implementability and rehostability of [CAIS85] from a Sun[7] UNIX system to a VAX[8] VMS system [RCAIS]. This prototype implementation was then extended by MITRE to address issues concerning a heterogeneous distributed system [DCAIS].

---

[7]Sun is a registered trademark of Sun Microsystems, Inc.

[8]VAX is a registered trademark of Digital Equipment Corporation.

2. THE CAIS NODE MODEL

The CAIS provides for the administration of the life cycle development and maintenance of embedded computer systems. Typically in an APSE, a number of users employ a variety of software tools to produce files of design, source and object code and other project information. Interfaces for the use of files, processes and directory spaces are given. Since it is desirable that tools be transportable among APSEs, common interfaces must be provided between the tools and the host system. In order to arrive at a coherent set of interfaces, the CAIS has been designed around an abstract model to support the life cycle development and maintenance of applications software which support embedded computer systems. This model is the CAIS node model.

The CAIS node model consists of entities (e.g., files, devices, directories) and associations between these entities. The model is depicted as a directed graph of nodes (nodes of the graph) and relationships (arcs of the graph). The nodes represent entities and can have contents and attributes. The relationships specify unidirectional associations from one node to another and can also have attributes. Attributes provide information about the node or relationship with which they are connected. Figure 4 shows the node model in the form of a schema. The node model is described in Section 4.3 of [1838].

2.1 Example Scenario

A user, Jones, is a member of a software engineering project of a large aerospace company and is responsible for developing a subprogram that works with radar information as part of a landing system. Within this company, the project that Jones belongs to is known as the Tracker project. Jones has previously organized his work within the CAIS database. In particular, he has organized several program units together for the Tracker project; one of these areas is for the landing system subproject and it contains the radar subprogram.

When any user logs on to this company's CAIS-based APSE, a command language interpreter (CLI) is started on behalf of that user. The code for the CLI and other user tools are organized together within the CAIS database. In addition, within the tools area of the CAIS database, several tools have been collected together that deal especially with the subproject that Jones is working on in the Tracker project of this company. One of the tools located here is a simulator.

On this particular day, Jones has logged in to a terminal (a CRT), the CLI has been started and some CLI commands from a file have been performed automatically. From the CLI, Jones starts a simulation job in the background and then begins to edit an Ada subprogram source file. Figure 5 is a depiction of the node model from the perspective of user Jones. Today, the primary focus of user Jones is with the Tracker project. Thus, the editing tool that Jones is using reflects this (via the arrow labeled "current_node" in Figure 6).

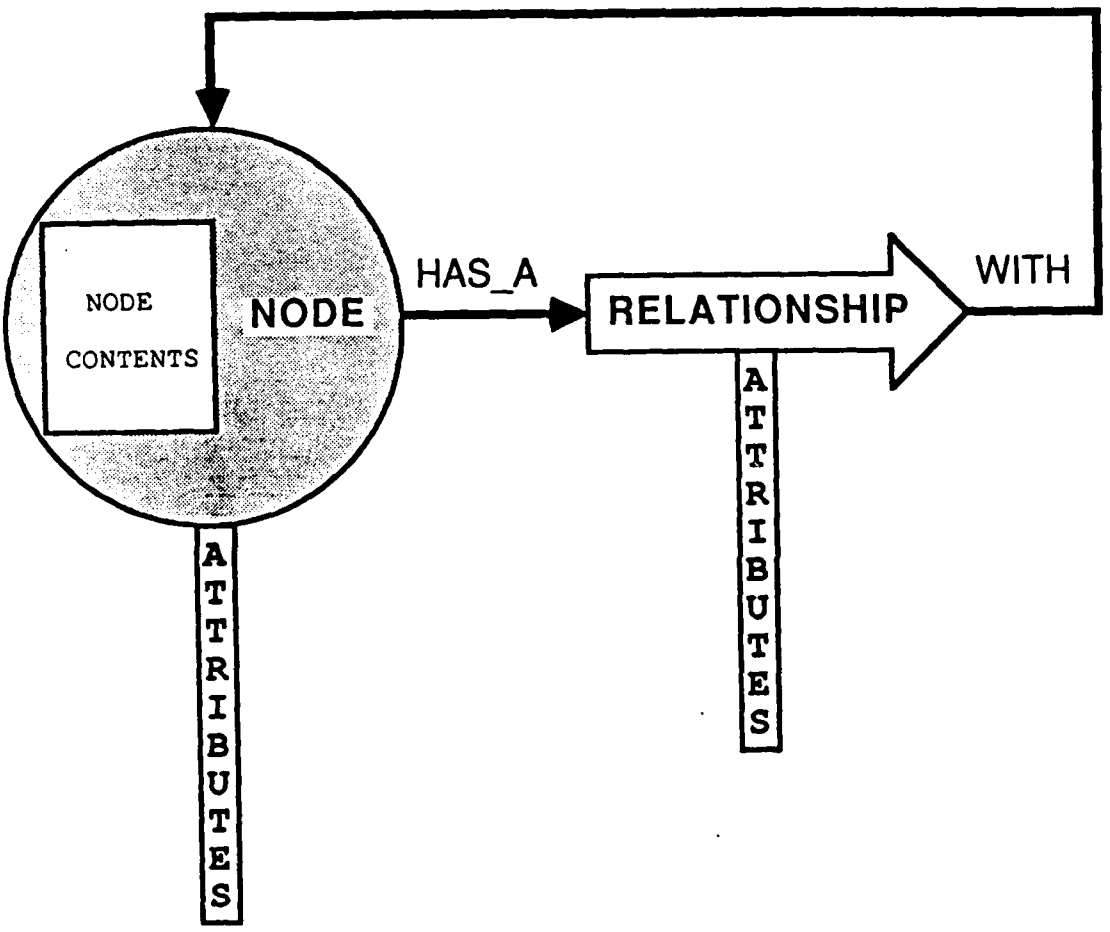


FIGURE 4. Node Model Schema

2.2 Nodes

The CAIS node model specifies three kinds of nodes: process, file and structural. The contents of process nodes represent the execution of Ada programs. File nodes represent Ada external files, such as data files, source code files or logical stores of information such as terminals. Structural nodes typically represent user collections of files, such as user directories.

The CAIS node model defines the concept of a system-level node, which represents the root of the graph spanned by the relationships between nodes. There is only one of these nodes. The CAIS does not provide facilities for operations on the system-level node. Figure 5 is a node model example showing process, file and structural nodes with the system-level node[9].

Nodes connected to the system-level node are called top-level nodes. In Figure 5, structural nodes identified by JONES and TOOLS are top-level user nodes. Top-level user nodes cannot be created or deleted with CAIS facilities. Such nodes represent the directories of users, which may be individuals, projects, services or other organizational entities that require access to APSE tools and files.

Devices in a programming environment, such as terminals and tape drives, are represented as file nodes associated with the system-level node. These file nodes are referred to as device nodes. Device nodes cannot be created or deleted with CAIS operations. The file node identified by CRT in Figure 5 is a device node.

Facilities are not provided in the CAIS to create and delete device nodes and top-level user nodes as explained above. Additional facilities must be provided by an implementation to support top-level nodes because these facilities are different on every system and thus it would be difficult to define a way that would be common across these diverse systems.

When a user enters an APSE implemented using the CAIS, a process node is created which forms the root of a tree of process nodes for that user. This node is referred to as a root process node. A root process node may represent, for instance, a command language interpreter. Nodes identified by CLI and SIMULATOR are root process nodes in Figure 5. A job is the tree of process nodes with the root process node as its root. Two jobs are shown in Figure 5: one consisting of the nodes identified by CLI and EDIT, and another consisting solely of the node identified by SIMULATOR. A user may have more than one job executing at any given time.

Within the node model, each process has a corresponding file node, known as a program node, whose contents is the executable image for that process. Each program node is the target of a secondary relationship of the predefined relation EXECUTABLE_IMAGE emanating from the process node (see Sections 2.3 and 4.4 of this guide). In Figure 5, for example, the nodes identified by CLI_CODE, EDIT_CODE, and SIMULATOR_CODE are the program nodes for the process nodes identified by CLI, EDIT and SIMULATOR, respectively (the secondary relationships of the EXECUTABLE_IMAGE relation are not shown in this figure).

---

[9]Note that in this figure as well as other figures in this guide which show portions of the node model, not all relationships are shown; only those relationships that are currently under discussion are depicted.

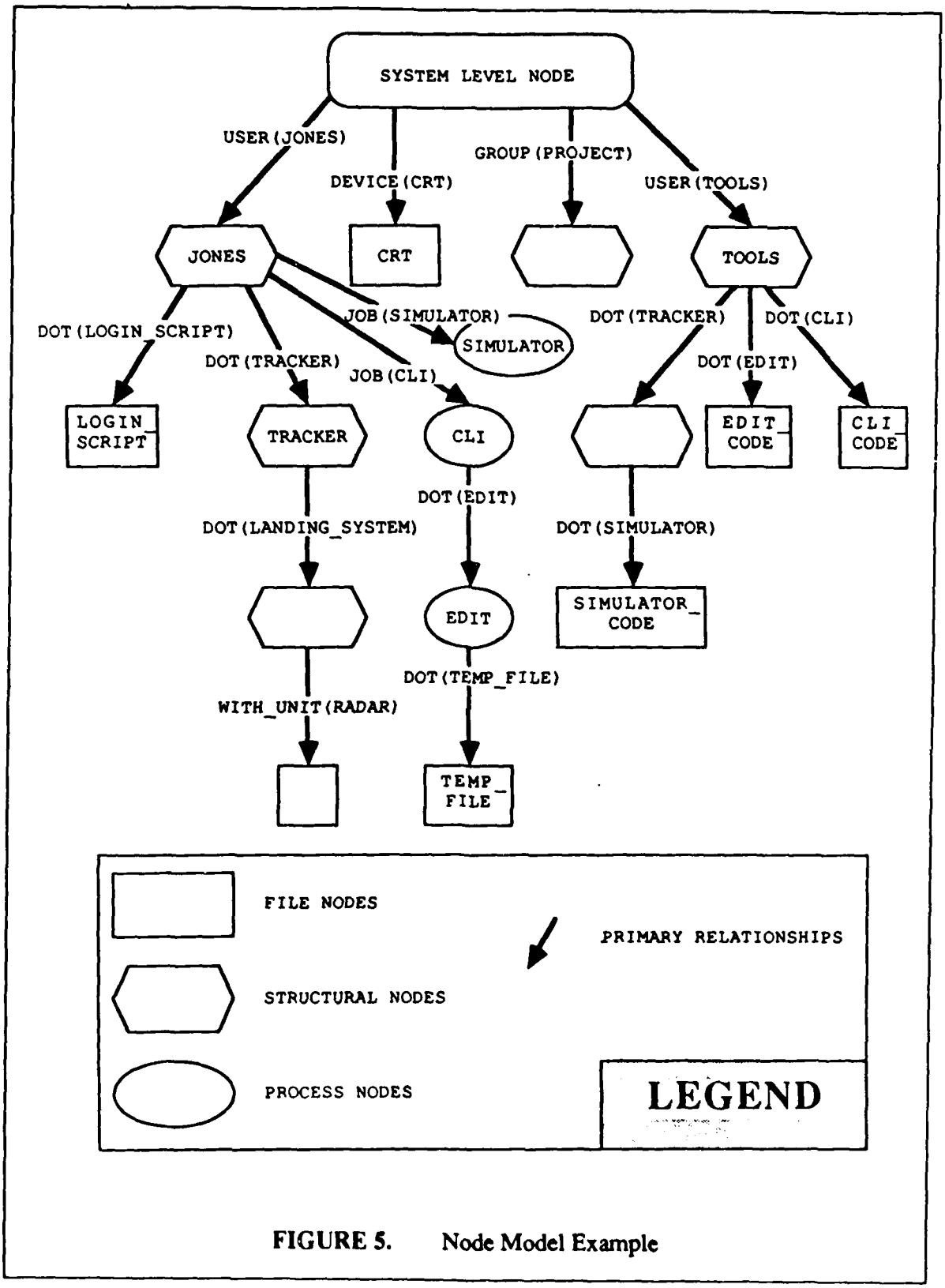


FIGURE 5. Node Model Example

2.3 Relationships and Relations

Nodes within an APSE are usually interrelated. Nodes within the node model are connected by unidirectional arcs called relationships. A relation is a set of relationships having the same name. A relationship is an instance of a relation, much like a set element is an instance of a set. Relationships are identified by their relation name together with a string called a relationship key. Relationships having the same relation name and emanating from the same node are differentiated by the use of relationship keys.

For example, in Figure 5, the nodes identified by CLI_CODE and EDIT_CODE are targets of relationships of the relation DOT. Since there is more than one relationship with the same relation name emanating from the node identified by TOOLS, the relationships must be differentiated by the use of relationship keys. In this example, the relationship keys are CLI and EDIT, respectively.

The CAIS provides two distinct categories of relationships: primary relationships and secondary relationships. When a node is created, a primary relationship is created between some existing node and the created node. The existing node is known as the parent of the created node. Accordingly, the created node is referred to as a child of the existing node. Each node is the target of exactly one primary relationship. Graph interconnections representing primary relationships are such that the graph forms a tree. The tree structure may be changed through the creation, deletion and renaming of primary relationships, but it always remains a tree. Secondary relationships are used to establish additional associations between nodes. Emanating from each child is a secondary relationship of the predefined relation PARENT that identified the child's parent. In Figure 5, for example, the node identified by JONES is the parent of the nodes identified by LOGIN_SCRIPT and TRACKER; similarly, the nodes identified by LOGIN_SCRIPT and TRACKER are children of the node identified by JONES.

The primary relationship to a node may be deleted, rendering the node unobtainable. If the primary relationship to a node has been deleted, the node may actually still exist in the implementation, but it cannot be accessed. Such nodes are not referred to as deleted nodes because the implementation controls when the node is physically deleted (e.g., expunged from storage). An operation is also provided to delete all of the primary relationships within a subtree, thus rendering all nodes in the subtree unobtainable. The concept of an unobtainable node is further explained in Section 3.1, Node Operations, of this guide.

Since objects in an environment often have many associations, the CAIS node model provides for nodes to be connected through secondary relationships. The structure formed by the secondary relationships is a directed graph. Secondary relationships to a node remain in existence even after the node is made unobtainable. These relationships remain in existence because it is possible that attributes of these relationships may still contain important information about the node. To eliminate these relationships, they must be specifically deleted. However, any attempt to access an unobtainable node (one whose primary relationship has been deleted) through the secondary relationship will result in an exception being raised.

The primary relationship to a node may be renamed. This operation deletes the primary relationship to the node and establishes a new primary relationship to that node from another specified node. The secondary relationship of the predefined relation PARENT to the

original parent is deleted and a new secondary relationship of the predefined relation PARENT is created to the new parent. All secondary relationships to the renamed node remain.

Nodes are accessed by establishing a path from a reference or current node; this is called navigation. Nodes may be accessed by specifying a path through the node model using primary and secondary relationships. Each path has a pathname which consists of the concatenation of the relation names and relationship keys used to move from the first node in the path to the last node in the path. Pathnames can be constructed of both primary and secondary relationships. 'USER(TOOLS)'DOT(TRACKER)'DOT(SIMULATOR) is an example of a pathname of the node identified by SIMULATOR_CODE from Figure 5, where USER and DOT are predefined relations that will be explained later. An alternate method of referring to a node is through a node handle, which is discussed in Section 3.1, Node Operations, in this guide.

The CAIS defines a number of predefined relations in the node model. User defined relations can also be established. Seven of the predefined relations are: PARENT, USER, DEVICE, JOB, CURRENT_JOB, CURRENT_NODE and CURRENT_USER. Figure 6 depicts a portion of the node model example from Figure 5 with relationships of some of these relations shown (all relationships for all relations are not shown in Figure 6). Secondary relationships of the PARENT relation emanate from each child to its parent. There are predefined relations USER and DEVICE which may be used for both primary and secondary relationships. Primary relationships of the relation USER emanate from the system-level node and identify user top-level nodes; secondary relationships of the relation USER emanate from process nodes and also identify user top-level nodes. Primary relationships of the relation DEVICE emanate from the system-level node and identify device nodes; secondary relationships of the relation DEVICE emanate from process nodes and also identify device nodes. Each root process node is identified by a primary relationship of the JOB relation from its parent. Additionally, each process node has secondary relationships of the CURRENT_JOB, CURRENT_NODE and CURRENT_USER relations emanating from it. These relationships are typically used as a context for the interpretation of pathnames. A relationship of the relation CURRENT_JOB always identifies the root process node of the subtree containing the process. A relationship of the relation CURRENT_NODE is typically used to identify the node that is the object of that process's actions; it defines part of the current environment for pathname interpretation. A relationship of the relation CURRENT_USER identifies the top-level user node for that process.

DOT is the default relation used when none is otherwise specified. In pathname syntax, 'DOT can be abbreviated to ".". The relationship key is then used to differentiate between other nodes sharing the DOT relation with the parent (i.e., from the previous pathname example, 'USER(TOOLS).TRACKER.SIMULATOR is the abbreviation). Relationships of the DOT relation must always have a relationship key.

Table I is a list of the predefined relations. For more information on relations, see Section 4.3 and Appendix A of [1838].

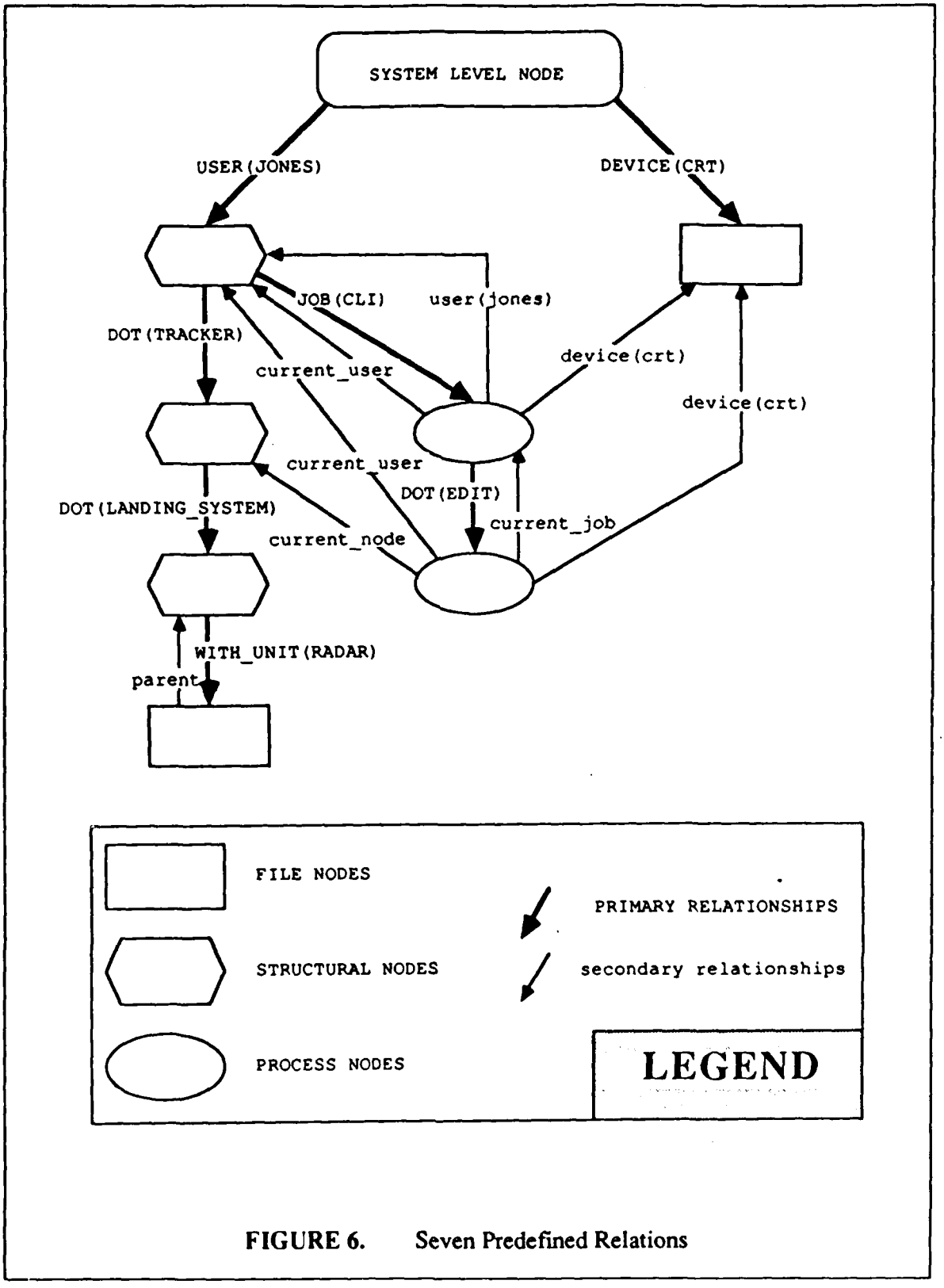


FIGURE 6. Seven Predefined Relations

TABLE I. Predefined Relations

Predefined Relation	Relationship Type	Source Node	Target Node
ACCESS	secondary	any	group
ADOPTED_ROLE	secondary	process	group
CURRENT_JOB	secondary	process	root process
CURRENT_NODE	secondary	process	any
CURRENT_USER	secondary	process	top-level user
DEFAULT_ROLE	secondary	top-level user	group
DEVICE	primary secondary	system-level process	device top-level device
DOT	primary or secondary	any	any
EXECUTABLE_IMAGE	secondary	process	file
GROUP	secondary primary	process system-level	top-level group top-level group
JOB	primary	top-level user	root process
MIMIC_FILE	secondary	mimic queue	coupled file
PARENT	secondary	any	any
POTENTIAL_MEMBER	secondary	group	group
USER	primary secondary	system-level process	top-level user top-level user

When "any" is indicated in this table, it is understood to mean "any node except the system-level node".

The relations ACCESS, ADOPTED_ROLE, DEFAULT_ROLE, GROUP and POTENTIAL_MEMBER are not discussed in this guide. These five relations are used in discretionary access control, which is only briefly mentioned in this guide.

2.4 Attributes

Information about nodes and relationships may be contained in the attributes of the nodes or the attributes of the relationships. Each attribute has a name and a value; the attribute value is a list. For example, each CAIS node has a predefined attribute TIME_CREATED which indicates the implementation-defined time of creation for that node.

There are several predefined attributes in the CAIS. Table II lists the predefined attributes and describes their applicability and values. More information on attributes can be found in Section 4.3 and Appendix A of [1838]. Figure 7 shows an example of several predefined node attributes and relationship attributes.

TABLE II. Predefined Attributes

Predefined Attribute	Applicable to	Values
ACCESS_METHOD	file node	SEQUENTIAL, DIRECT, TEXT
CURRENT_FILE_SIZE	secondary storage file node	value >= 0
CURRENT_QUEUE_SIZE	nonsynchronous queue file node	value >= 0
CURRENT_STATUS	process node	READY, SUSPENDED, ABORTED, TERMINATED
DEVICE_KIND	device file node	SCROLL_TERMINAL, PAGE_TERMINAL, FORM_TERMINAL, MAGNETIC_TAPE_DRIVE
FILE_KIND	file node	SECONDARY_STORAGE, QUEUE, DEVICE
GRANT	ACCESS relationships	access right value
HIGHEST_CLASSIFICATION	file node	implementation-defined
INHERITABLE	all relationships	TRUE, FALSE
IO_UNIT_COUNT	process node	integer >= 0
LOWEST_CLASSIFICATION	file node	implementation-defined
MACHINE_TIME	process node	value >= 0
MAXIMUM_FILE_SIZE	secondary storage file node	value >= 0
MAXIMUM_QUEUE_SIZE	nonsynchronous queue file node	value >= 0
NODE_KIND	all relationships	STRUCTURAL, PROCESS, FILE
OBJECT_CLASSIFICATION	all nodes	implementation-defined
OPEN_NODE_HANDLE_COUNT	process node	value >= 0
PARAMETERS	process node	tool-defined list
PROCESS_SIZE	process node	value >= 0
QUEUE_KIND	queue file node	SYNCHRONOUS_SOLO, NONSYNCHRONOUS_SOLO, NONSYNCHRONOUS_COPY, NONSYNCHRONOUS_MIMIC
RESULTS	process node	tool-defined
SUBJECT_CLASSIFICATION	all nodes	implementation-defined
TIME_ATTRIBUTE_WRITTEN	all nodes	implementation-defined time
TIME_CONTENTS_WRITTEN	file node	implementation-defined time
TIME_CREATED	all nodes	implementation-defined time
TIME_FINISHED	process node	implementation-defined time
TIME_RELATIONSHIP_WRITTEN	all nodes	implementation-defined time

TABLE II. Predefined Attributes -- Continued.		
Predefined Attribute	Applicable to	Values
TIME_STARTED	process node	implementation-defined time

Attributes may also be defined by the user. User-defined attributes are used for describing additional characteristics of nodes and relationships. For example, user-defined attributes may be useful for describing the number of pages in a document. An attribute NUMBER_OF_PAGES, for example, could be defined for each file node that contains a document to specify the length, in pages, of that document.

2.5 Security and Access Control Provisions

There are two kinds of access controls defined in the CAIS document: discretionary access controls and mandatory access controls.

Discretionary access controls limit the authorized access of process nodes to other nodes. Along with certain relations that determine which nodes a process can act upon, the concept of a role, which represents access rights, establishes discretionary access controls. For more detailed information on access controls, see Section 4.4 of [1838].

Some APSEs may be required to operate in multi-level secure environments. The CAIS supports mandatory access controls. These controls remain consistent with the security criteria established in [TCSEC] and define neither a particular security model nor policy. CAIS mandatory access control provides a labeling mechanism via node attributes. By classifying nodes, mandatory access controls limit the use of the three operations of reading, writing, or reading and writing. Such classifications may be hierarchical (e.g., the conventional government UNCLASSIFIED, CONFIDENTIAL, SECRET and TOP SECRET hierarchy of classifications), non-hierarchical (e.g., compartmented), or a combination of both.

2.6 CAIS Operations

The interfaces specified in the CAIS may be divided into three main groups:

a. General node management operations.
b. Process node operations.
c. Input and Output operations.

These operations are described in the following chapters of this guide. The CAIS also includes a number of utility packages which are not covered in this guide.

a. Section 5.4 of [1838] discusses list management of the CAIS.

b. Section 5.5 of [1838] defines the package CAIS_STANDARD which contains certain scalar types predefined in the CAIS; this package is provided in order to make these types reasonably independent of any predefined types in the Ada language, whose characteristics may vary among compilers.

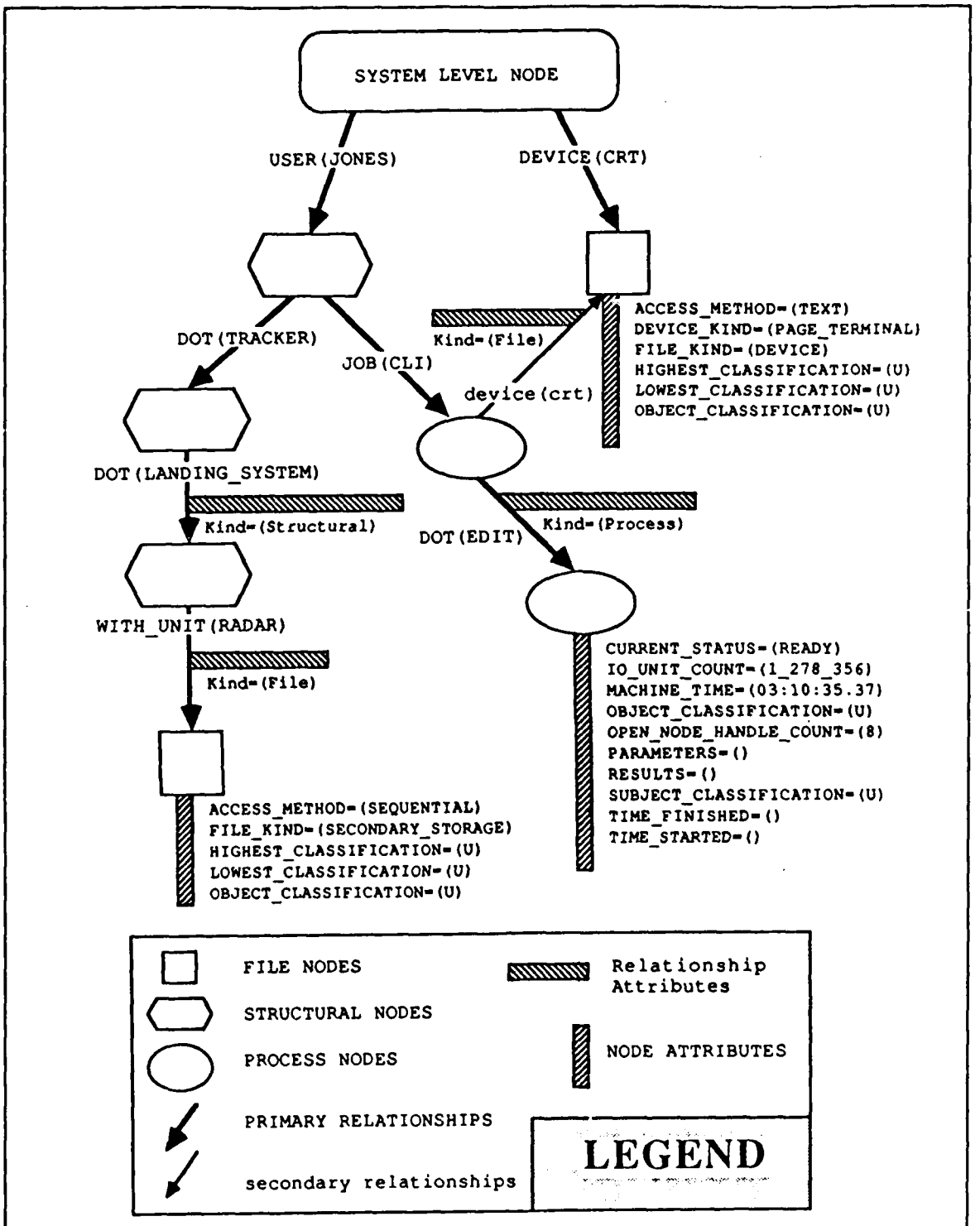


FIGURE 7. Some Predefined Attributes

c. Section 5.6 of [1838] defines the package CAIS_CALENDAR which provides facilities for accessing a system clock and interpreting its values.

d. Section 5.7 of [1838] defines the package CAIS_PRAGMATICS. Pragmatics are constraints imposed by an implementation that are not defined by the syntax and semantics of the CAIS. The minimum capacities that must be supported by a system that supports the CAIS are provided in this package.

3. GENERAL NODE MANAGEMENT

General node management operations involve node operations, attribute operations and discretionary access control operations. Node operations include identifying, creating, duplicating and deleting nodes and managing relationships between nodes. Attribute operations include creating, deleting and modifying attributes on nodes and relationships. Discretionary access control operations include setting, deleting and examining access control information. Node management allows for the management of file, process and structural nodes in the CAIS node model.

3.1 Node Operations

Once a particular node has been identified by traversing a path, the pathname need not be re-evaluated. Instead, a node handle may be used. A node handle is a unique identification of a node and may be used to identify any node in the graph. The use of a node handle avoids repeatedly specifying long path traversals to a node which is often accessed. This handle can then serve as a convenient reference for that node. Node management operations frequently use node handles. To address a node using a node handle, a node handle must first be opened to that node. The open operation establishes the node handle and associates a node handle with the specified node. An open node handle is said to track the node it identifies, meaning that the node handle is guaranteed to refer to the same node, regardless of changes in any relationships. Sufficient access rights (e.g., to read or write the contents of a node) are necessary to successfully open a node handle. Moreover, if a node handle is closed, the node handle no longer refers to the given node.

The structure of the node model can be altered by the creation of nodes and relationships and by the deletion of relationships. The operations for creating nodes are provided in the CAIS by a variety of node and subtree creation procedures. Interfaces for these operations are discussed in Section 5.1.2 (for general node management) of [1838]. Structural nodes are created with the interfaces described in Section 5.1.5 of [1838]. Section 5.2.2 of [1838] describes the interfaces with which process nodes can be created and deleted. The interfaces described in Section 5.3 of [1838] can be used for the creation and deletion of file nodes (for input and output).

The primary relationship to a node may be deleted. In many ways, the node itself may be thought of as having been "deleted", although the CAIS leaves it up to the implementation whether or not the physical node still exists. A node whose primary relationship to it has been deleted is said to be unobtainable. Secondary relationships identifying unobtainable nodes, however, remain until explicitly deleted by relationship deletion operations. Open node handles to a node that has become unobtainable still refer to that node. Attempts to access the contents or attributes of an unobtainable node using secondary relationships or open node handles will result in an exception being raised. An exception will also be raised if an attempt is made to access a relationship emanating from an unobtainable node. The attributes of relationships targeting the node may still be examined, though.

File and structural nodes may be copied. All secondary relationships emanating from the node are also copied. Secondary relationships to the original node are not copied.

An entire subtree of file and/or structural nodes may also be copied. Secondary relationships between two nodes in the subtree, both of which are copied, are also recreated between the newly copied nodes. Secondary relationships emanating from copied nodes to nodes outside the subtree are copied, too. However, secondary relationships emanating from nodes outside the subtree to nodes inside the subtree are unaffected. Figure 8 shows an example of copying a subtree of nodes.

Renaming a file node or structural node (effectively changing the parent for that node) deletes the primary relationship to the node and installs a new primary relationship from another node, thus establishing a new secondary relationship of the PARENT relation to the renamed node. In Figure 5, for instance, the file node identified by TEMP_FILE may be renamed to have a new parent such as the structural node identified by TRACKER, thus making a permanent file out of a temporary file. Secondary relationships to the renamed node remain intact.

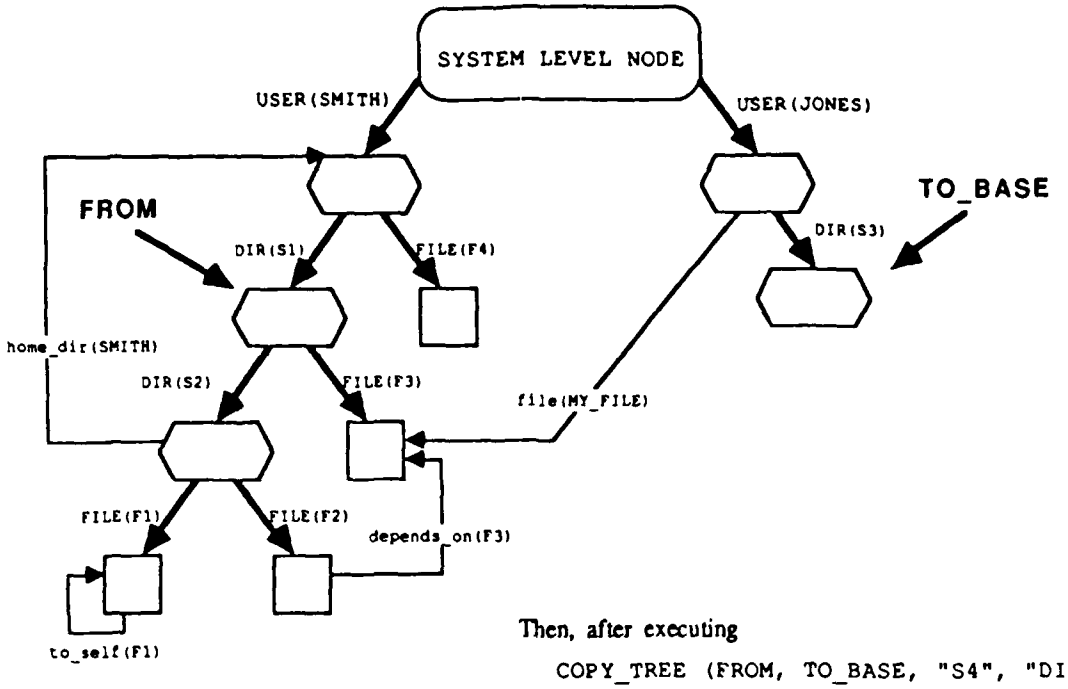
Operations are also available in the CAIS to identify all those nodes which emanate from a node and match a given relation name and relationship key pattern (called a relationship key descriptor). This may be useful, for instance, for finding all the relationships of a particular relation emanating from a node or for finding all the primary relationships emanating from a node. Interfaces for these operations are discussed in Section 5.1.2 of [1838].

3.2 Attribute Operations

Several interfaces exist for the manipulation of node attributes and relationship attributes. Attributes may be created and deleted. The values of attributes may also be examined and changed. Predefined attributes, however, may not be altered. The capability is also given to iterate through attributes of a specified node or relationship. Interfaces for these operations are discussed in Section 5.1.3 of [1838].

3.3 Discretionary Access Operations

As the number, nature and function of the tools involved in the APSE changes, discretionary access control may also change in the node model (so that, for example, individual projects may be protected). The CAIS provides mechanisms for changing discretionary access controls. Interfaces for these operations are discussed in Section 5.1.4 of [1838].



Then, after executing

```
COPY_TREE (FROM, TO_BASE, "S4", "DIR")
```

the node model becomes:

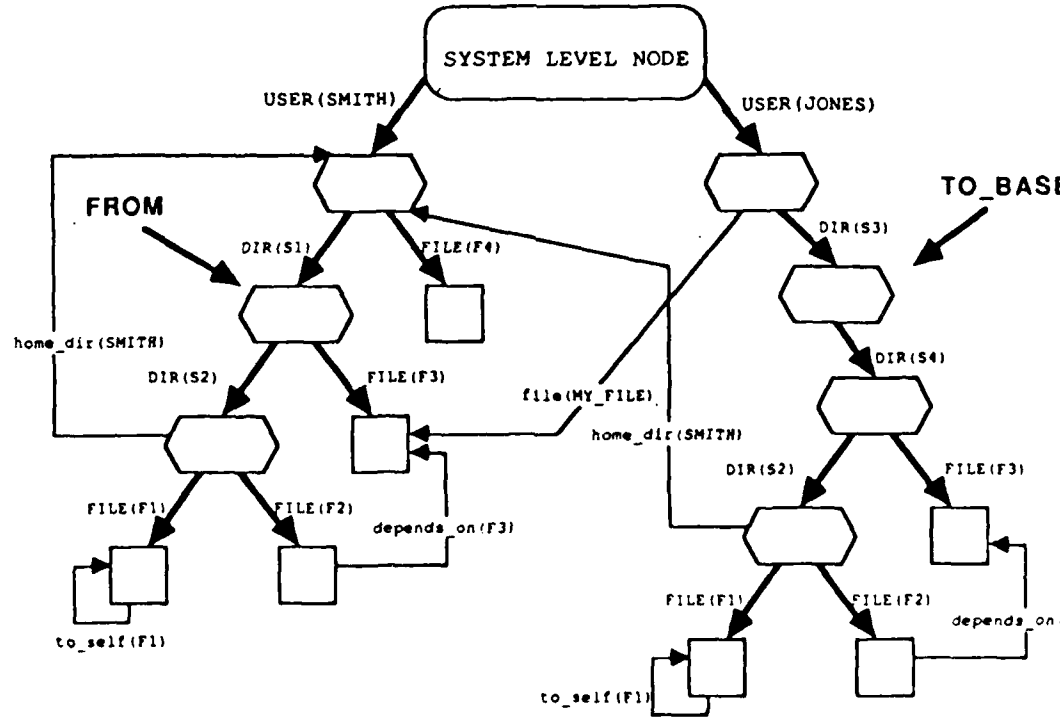


FIGURE 8. Copying a Subtree of Nodes

4. CAIS PROCESS NODES

An executing Ada program is a linked set of Ada tasks and Ada subprograms. The Ada Language Reference Manual [1815A] addresses only the functions available within a single program. There are requirements in APSEs for programs that are separately compiled and linked to interact with one another.

Figure 9 depicts the scope of the Ada program interaction. The individual Ada programs (programs A, B and C) each contain a number of subprograms and tasks which cannot interact with other executing programs. The scope of the CAIS definition encompasses the ability of several Ada programs to interact in a software system. In the CAIS, process nodes are used to represent each executing Ada program. In Figure 9, programs A, B and C could be represented as CAIS process nodes.

The CAIS recognizes that each executing Ada program is obtained from a file node containing the program's executable image. Several processes may be running simultaneously from this single executable image. Each process would perform the same functions as the other, except each would execute in a different context from the others. An example of such a situation is several APSE users performing compilations simultaneously. An important restriction on process nodes is that they cannot be copied or renamed (attached to a new parent).

Section 5.2 of [1838] describes the execution of Ada programs as represented by CAIS processes and the facilities provided by the CAIS for initiating and controlling processes.

The major events in the life of a process are:

a. Initiation.
b. Running, which may include suspension or resumption.
c. Termination or abortion.

Initiation creates a process node and begins running the process. After a process begins to run, it may be suspended. A suspended process may be caused to resume running. When a process completes running normally, it is said to terminate. However, a process may instead be aborted, which permanently stops the process from running. A state table diagram showing each of these process node states and transitions is given in Figure 10. Since processes are part of the CAIS node model, process nodes may be identified for the purpose of obtaining information about the process.

Interfaces that deal with processes are discussed in Section 5.2 of [1838].

4.1 Process Initiation

Initiation creates a process node and begins running the process.

When a user enters an APSE implemented using the CAIS, a process node is created which forms the root of a tree of process nodes for that user. This node is referred to as a root

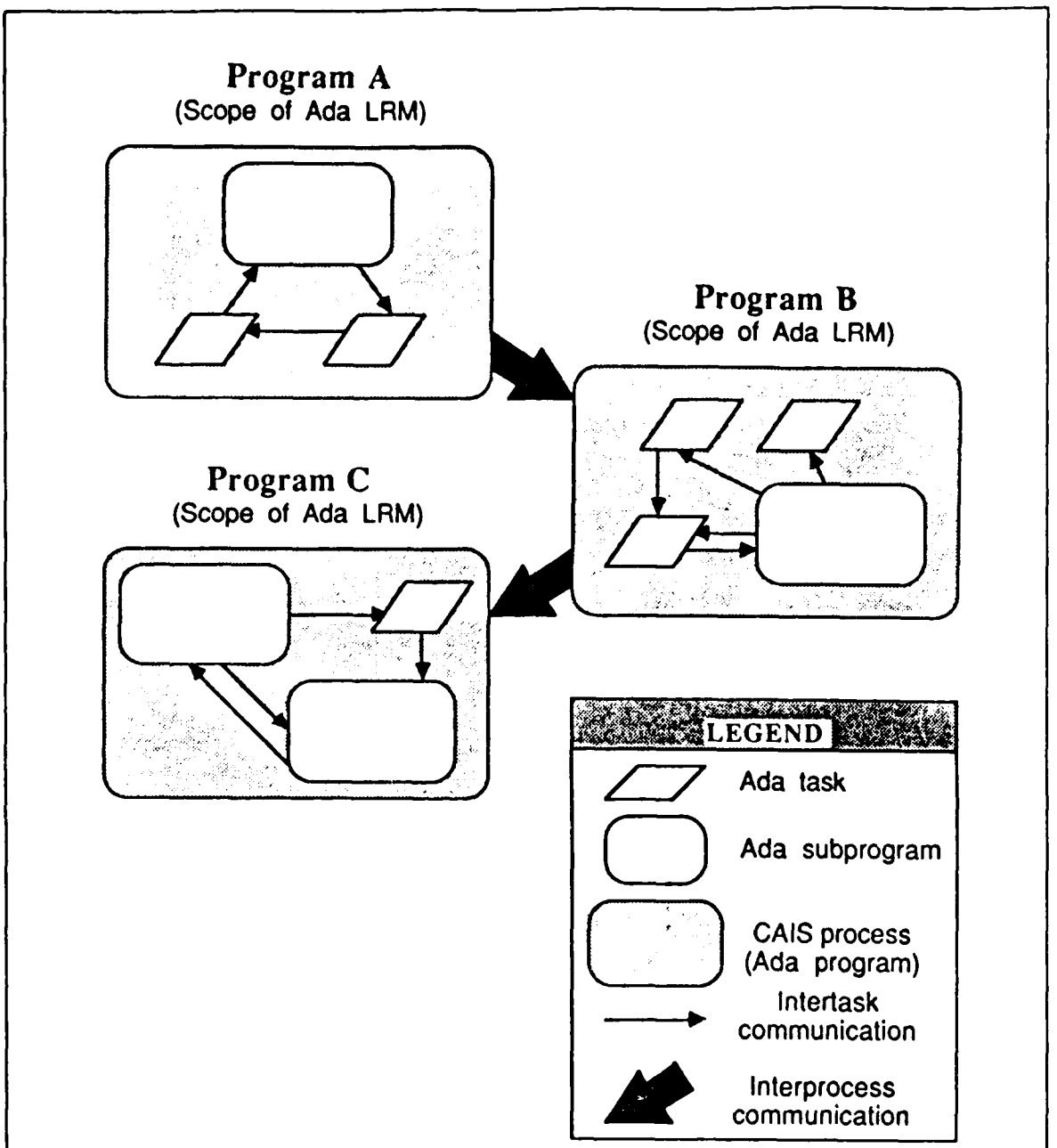


FIGURE 9. A Process is an Ada Program

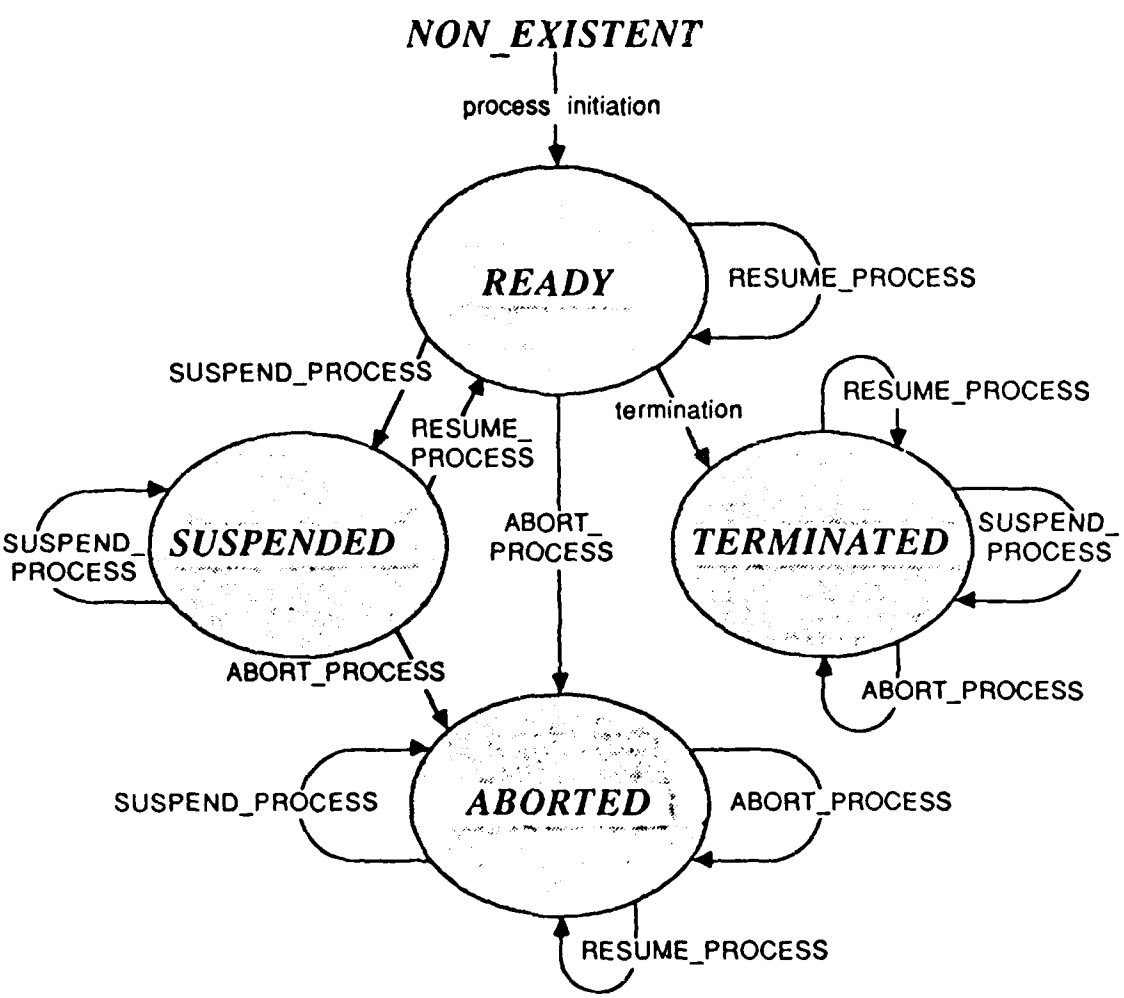


FIGURE 10. Process States

process node. Nodes identified by CLI and SIMULATOR are root process nodes in Figure 5. A job is the tree of process nodes with the root process node as its root. Two jobs are shown in Figure 5: one consisting of the nodes identified by CLI and EDIT, and another consisting solely of the node identified by SIMULATOR. A user may have more than one job executing at any given time.

All process nodes in a job form a tree. An initiating process is known as the parent process, while the initiated process is known as the child process. Except for the root process node, the parent of a process node is always another process node. For any process, a process tree consists of that process plus all of the processes which are descendants of that process.

A new job is created each time the user enters the APSE. The user may have more than one job if, for instance, he logs on at more than one terminal at one time. In addition, new jobs may also be explicitly created by other running processes.

There are three ways in which a process can initiate another process. These methods cause a process node to be created and a node handle to be opened.

The first method is referred to as invocation. The calling sequence to this interface does not return control to the task or subprogram of the invoking process until the child process ceases to run. Note, however, that any parallel tasks of the invoking process can continue to run.

The second method is referred to as spawning. A process can spawn many processes. For instance, from the CLI process, a user might spawn a simulator process, a compiler process (both of which do not need interactive inputs) and invoke an editor process. All three of these processes can be connected directly to the CLI process via primary relationships and run in parallel. The calling sequence to this interface returns control to the spawning process once the child process begins to run. The initiating process and the initiated process run in parallel, and within these processes, their tasks run in parallel.

Creating a new job is the third method of initiating a process. Creating a new job allows a user to create other root process nodes without logging on. This can be used to begin the execution of batch (background) processes. These processes run in parallel with other root process nodes and can themselves initiate other processes. Control is returned to the initiating process after the new job is created.

4.2 Process Suspension or Resumption

A running process may be suspended either by itself or by another process using CAIS operations, causing the process to stop running until another process causes it to resume running. A suspended process will continue running when it is the object of a resume operation. A process may be suspended (or resumed) even though its parent process is not suspended (or resumed).

4.3 Process Termination or Abortion

A process terminates when the Ada program it represents stops running normally. When a process is terminated, all of the processes in its process tree that are not terminated are aborted.

Any input and output files and node handles that are open are closed when a process terminates.

A process stops running when it is aborted. No further actions by the aborted process may occur. An aborted process may not later resume running. When a process is aborted, all of the processes in its process tree are likewise aborted. Thus, if a root process node is aborted, all of the other processes in the job are likewise aborted. In Figure 5, for example, if the process identified by CLI is aborted, the process identified by EDIT is also aborted. Similarly, if a process node is deleted while the process is running, the process is aborted. A process may be aborted even though its parent process is not aborted.

When a process terminates or aborts, the process node continues to exist until explicitly deleted, retaining useful attributes about the results and status of the execution.

4.4 Process Relationships

When a process node is created, a standard input file node, standard output file node and standard error messages file node are assigned as default files for input, output and error output for the process. These file nodes are identified by secondary relationships of the relations STANDARD_INPUT, STANDARD_OUTPUT and STANDARD_ERROR, respectively, at the beginning of a program's execution and remain as long as the program is executing.

Examples of these relationships are shown in Figure 11. The node identified by CLI, when created, had all three of these relations identifying a device node, the node identified by CRT. The file node identified by LOG_FILE later became the target of the relationship of the predefined relation STANDARD_ERROR. When the process node identified by EDIT was created, all three of the relationships identified the node identified by CRT. A relationship of the relation STANDARD_ERROR was later targeted to the node identified by LOG_FILE and a relationship of the relation STANDARD_INPUT was targeted to the node identified by DATA_FILE. The node identified by SIMULATOR still has the three original relationships; all of them emanate from the node identified by SIMULATOR and are targeted to the node identified by CRT.

Secondary relationships of the predefined relations CURRENT_JOB, CURRENT_NODE, CURRENT_USER, DEVICE and USER are also created when a process node is created. These were explained in Section 2.3 starting on page 14 (these relationships are not shown in Figure 11). In addition, a secondary relationship of the predefined relation EXECUTABLE_IMAGE is created to point to the program node for the process (this is only shown for the node identified by EDIT).

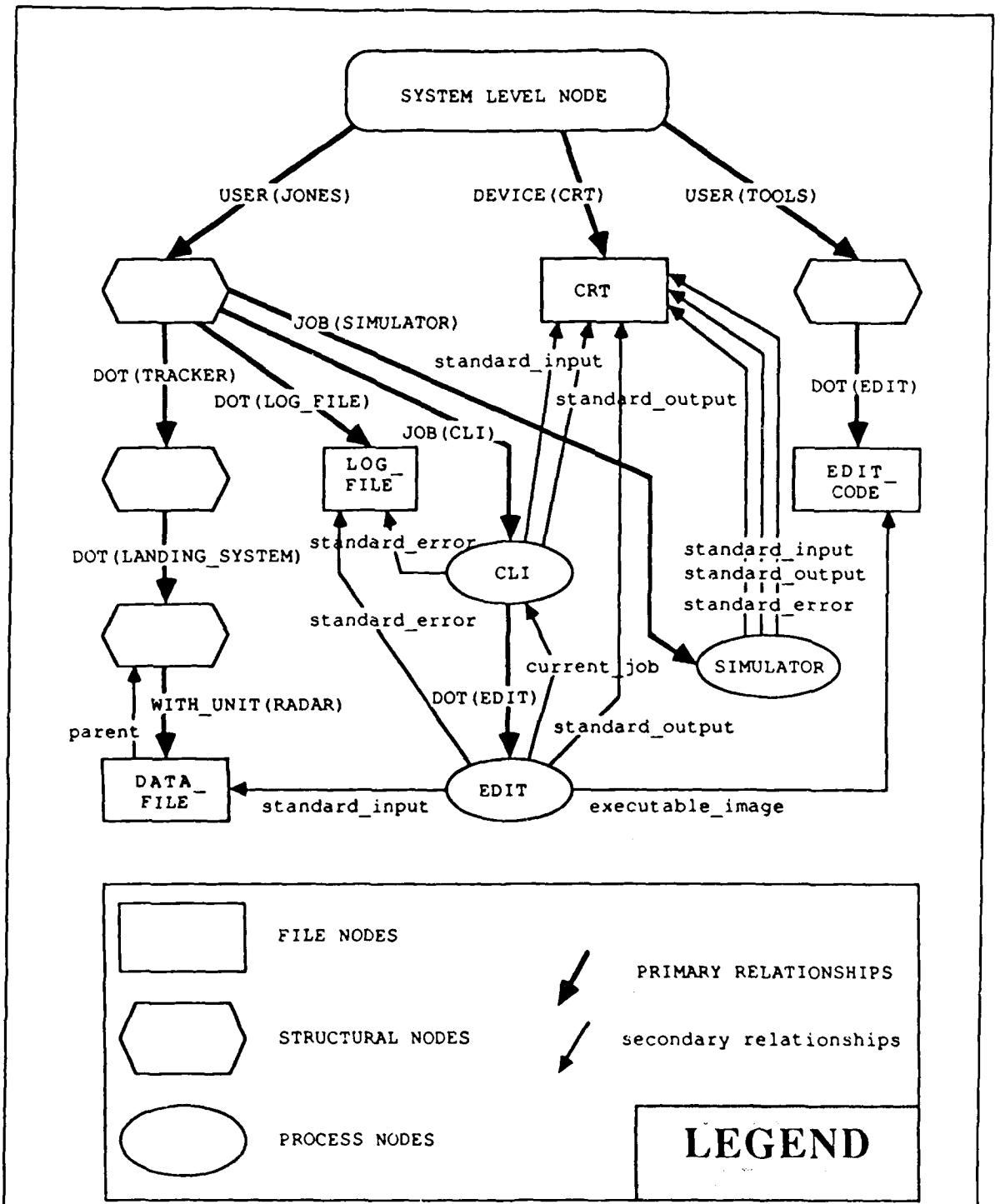


FIGURE 11. Input and Output Predefined Relations

4.5 Process Attributes

Each process node has several predefined attributes. Figure 7 shows an example with several of these attributes.

One of the attributes of the process node, CURRENT_STATUS, is used to determine whether the process is running, terminated, or aborted. This attribute may be checked at any time to determine the state of the process. This may be useful when it is necessary to know whether a process has been aborted or whether it is still running, but blocked.

Other information pertaining to a process can be obtained via attributes. IO_UNIT_COUNT is the number of input and output operations that a process has performed and OPEN_NODE_HANDLE_COUNT is the number of node handles currently open by the process. PROCESS_SIZE indicates the current size of a process, i.e., the amount of memory it is currently using. The attributes TIME_STARTED and TIME_FINISHED indicate the time that a process started and finished, respectively.

Each process node has a predefined attribute PARAMETERS which is a list of parameters given to the process by another process, e.g., a command line interpreter, when it was initiated.

Each process node has a predefined attribute RESULTS which is used to store a list of the results of the process. At any time when a process is running, it may store results in this list. The results of that process may be obtained by another process at any time while the original process is running or after the original process has ceased to run as long as the process node still exists.

5. CAIS INPUT AND OUTPUT

CAIS input and output operations incorporate input and output of the Ada language and also provide some new input and output capabilities.

CAIS input and output operations are used to transfer data to and from CAIS file nodes. There are three kinds of file nodes: secondary storage, queue and device. Secondary storage file nodes represent files such as disk files. Queue file nodes represent temporary stores of information typically used in operating systems for interprocess communication. The CAIS predefines four kinds of device file nodes: magnetic tape drives and scroll, page and form terminals. Magnetic tape drive file nodes represent magnetic tape drives. A terminal file node represents an interactive terminal. These file nodes are explained in more detail in the following sections of this guide. Section 5.3 of [1838] discusses the interfaces applicable to CAIS input and output.

5.1 Secondary Storage

A secondary storage file in the CAIS represents files such as disk files. The predefined node attributes CURRENT_FILE_SIZE and MAXIMUM_FILE_SIZE indicate the current and maximum sizes, respectively, of the secondary storage file.

5.2 Queues

Queues are files of information stored in a first-in first-out order and are primarily used for interprocess communication. Interfaces for queues are described in Section 5.3.7 of [1838]. Three categories of queues are used by the CAIS: solo queues, copy queues, and mimic queues.

Solo queues are the simplest form of a queue, allowing one or more processes to write to a queue while one or more processes read from the queue. Each item of information in the queue may only be read once. Processes typically send information to one another through solo queues. Figure 12 depicts a typical solo queue, where the process nodes identified by WRITER1 and WRITER2 are writing to the queue, and the process nodes identified by READER1, READER2 and READER3 are reading from the queue.

Figure 13 shows an example of how a solo queue can be used as a pipe between two processes. The process identified by PROCESS generates some unsorted output that is written to the solo queue identified by PIPE so that the process identified by SORT may read it. This process sorts the information it reads from the solo queue and sends it to the device node identified by CRT.

A copy queue is initialized with a copy of the contents from another file at the time of queue creation. An example of a copy queue is shown in Figure 14. This type of queue is useful for allowing one process to partially read a file and another process to read the rest of the file, as in batch processing. Copy queues are also useful for treating a secondary storage file node as a queue without disturbing the contents of the file.

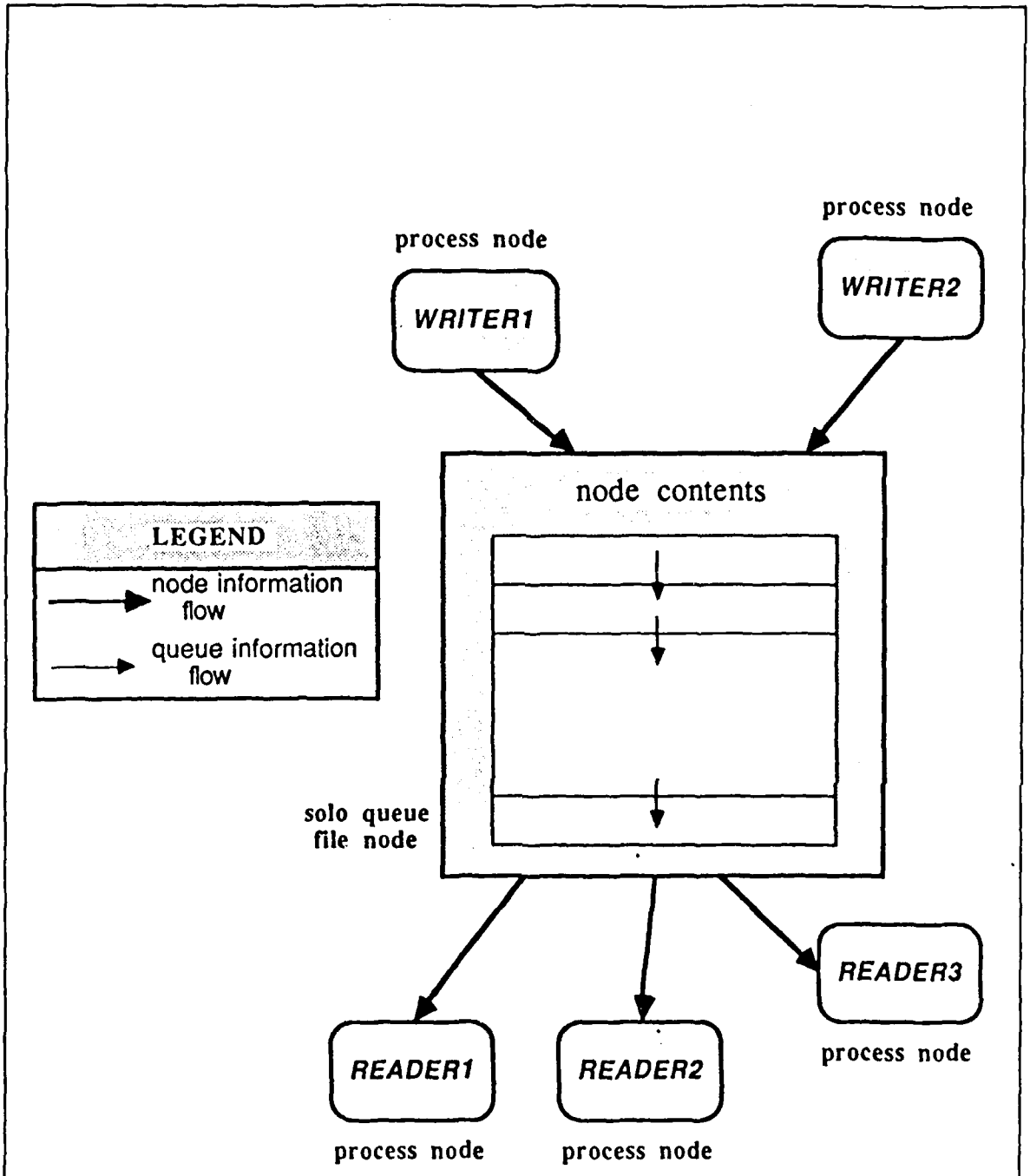


FIGURE 12. Solo Queues

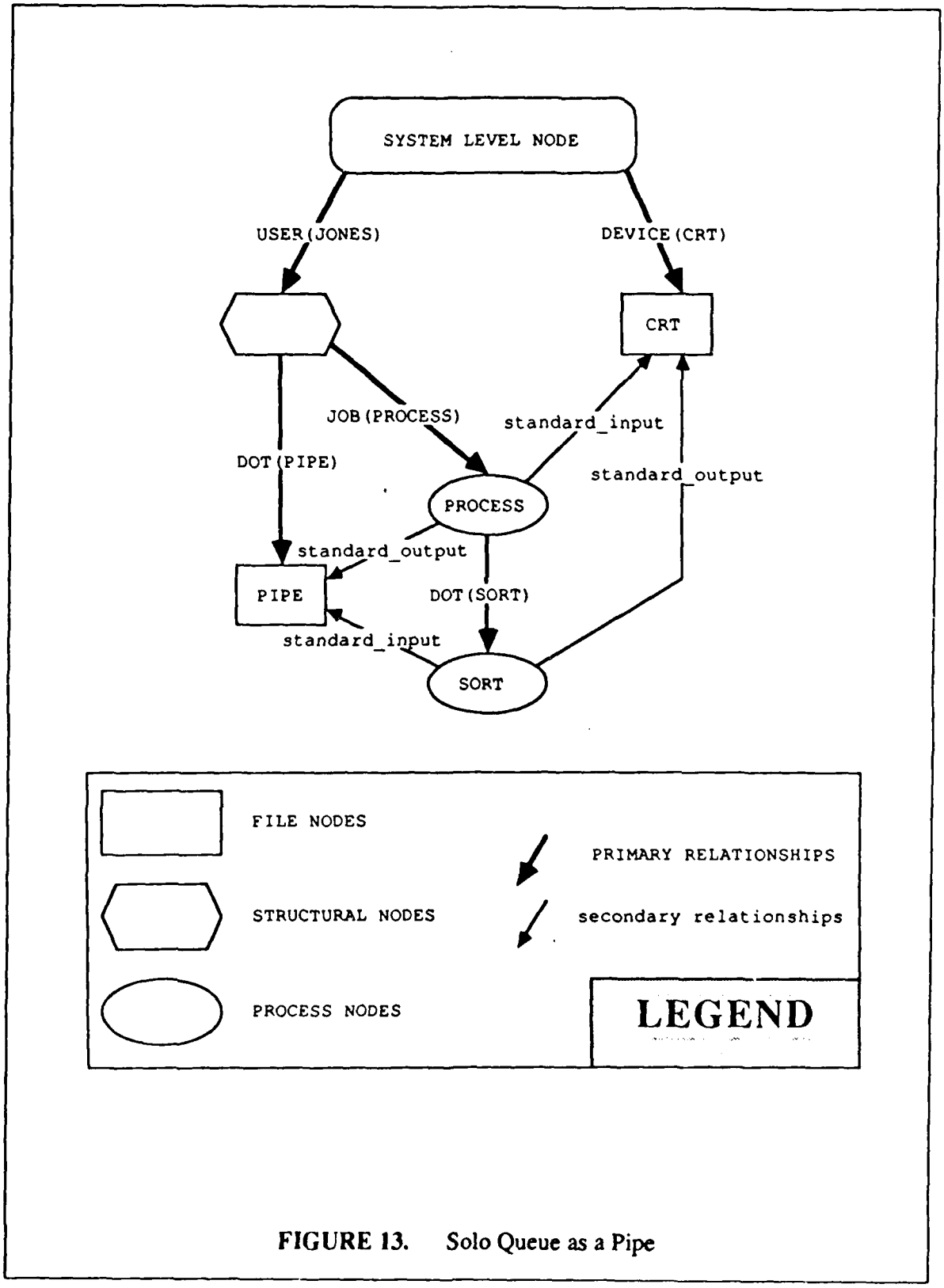


FIGURE 13. Solo Queue as a Pipe

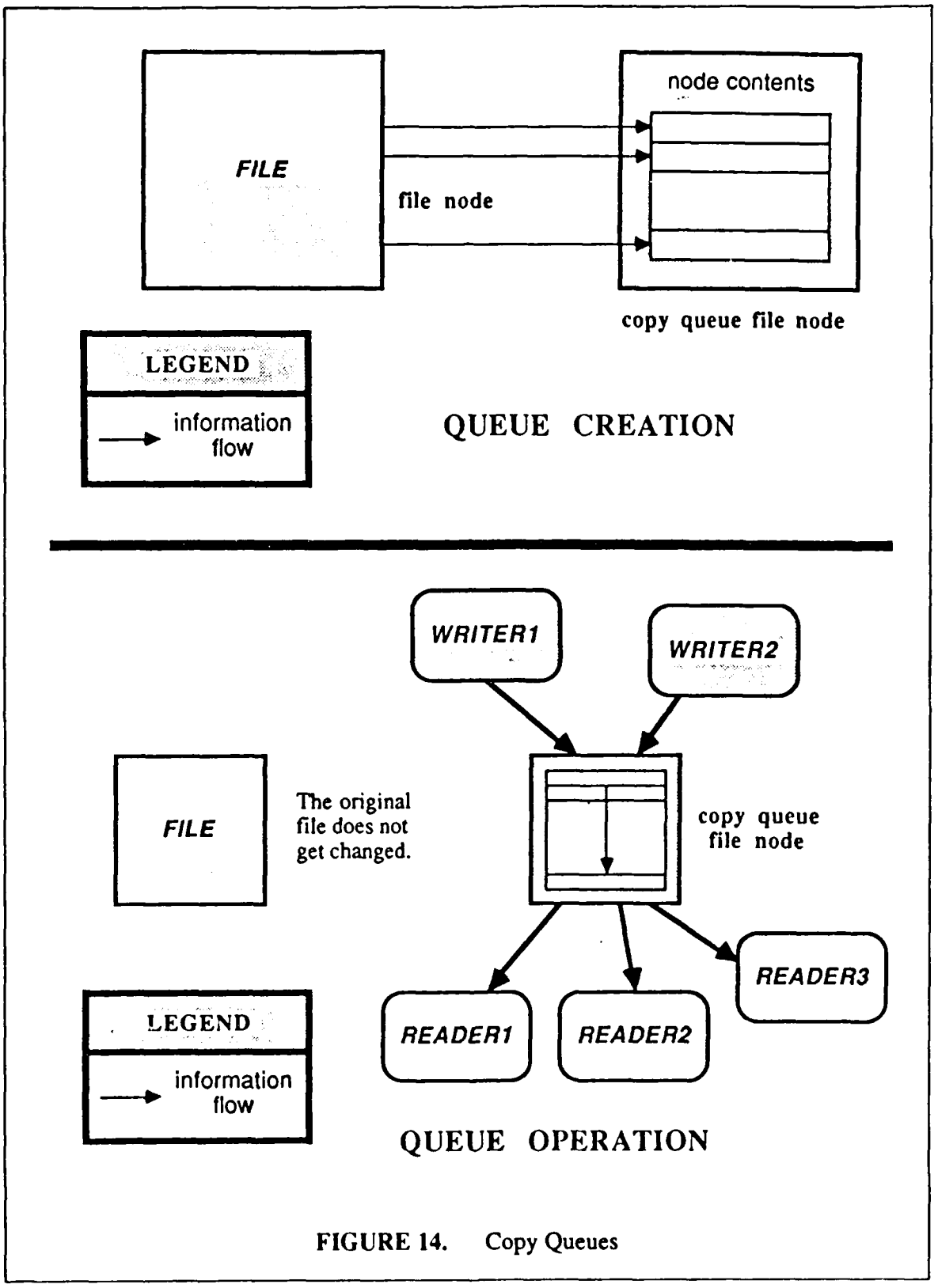


FIGURE 14. Copy Queues

A mimic queue is initialized like a copy queue, i.e., with a copy of the contents of another file at the time of queue creation. A secondary relationship of the predefined relation MIMIC_FILE points from the mimic queue file node to the file node from which the mimic queue is initialized. This file is called the coupled file. If information is written to a mimic queue, it is also appended to the coupled file. The effect on the mimic queue of modifications made to its coupled file is not defined by the CAIS. Mimic queues allow multiple processes to append information to secondary storage files without having to rewrite the files. This allows a form of logging. Figure 15 illustrates a mimic queue.

The examples of queues described above are examples of nonsynchronous queues. A nonsynchronous queue is a queue where several write operations to the queue may occur before any read operations occur.

The CAIS also defines a synchronous queue; only a solo queue can be a synchronous queue. A synchronous queue is one for which a write operation on the queue is not completed until a corresponding read operation on the same queue is completed, thus synchronizing the write operation with the read operation. A synchronous solo queue can be used to synchronize the operation of two cooperating processes.

5.3 Devices

A device file in the CAIS represents a device. The CAIS predefines magnetic tape drive files and three kinds of terminal files: scroll, page and form terminals.

5.3.1 Magnetic Tape Drives

Typically, an APSE includes a database of information upon which the tools in the APSE operate. In order for this database to be used on a different host system, there should be convenient mechanisms to transfer data files from one system to another. Minimal tape input and output operations are provided by the CAIS to allow files to be transported from one CAIS implementation to another.

ANSI tapes are handled by CAIS input and output. Tapes that are compliant with the ANSI and ISO standards [ANSI 73a; ANSI 73b; ANSI 76; ANSI 78; ISO 76a; ISO 76b; ISO 84] can be operated upon by the CAIS interfaces. Interfaces for magnetic tape drive files are described in Section 5.3.11 of [1838].

5.3.2 Terminals

Input and output operations are currently provided by the CAIS for only three kinds of terminals: scroll terminals, page terminals and form terminals. Scroll terminals, such as teletypes, use the keyboard as an input device and an output area as an output device and may input or output only one line at a time. Thus, each line is scrolled through. Interfaces for scroll terminals are described in Section 5.3.8 of [1838]. A page terminal is a non-intelligent CRT terminal which displays a "page" of lines at one time. Individual character positions may be addressed on a page terminal. Interfaces for page terminals are described in Section 5.3.9 of [1838]. Form terminals are used to display a "form" or menu to the display screen. After the user has modified the form, the modified form is read by the terminal controller. Interfaces for form terminals are described in Section 5.3.10 of [1838].

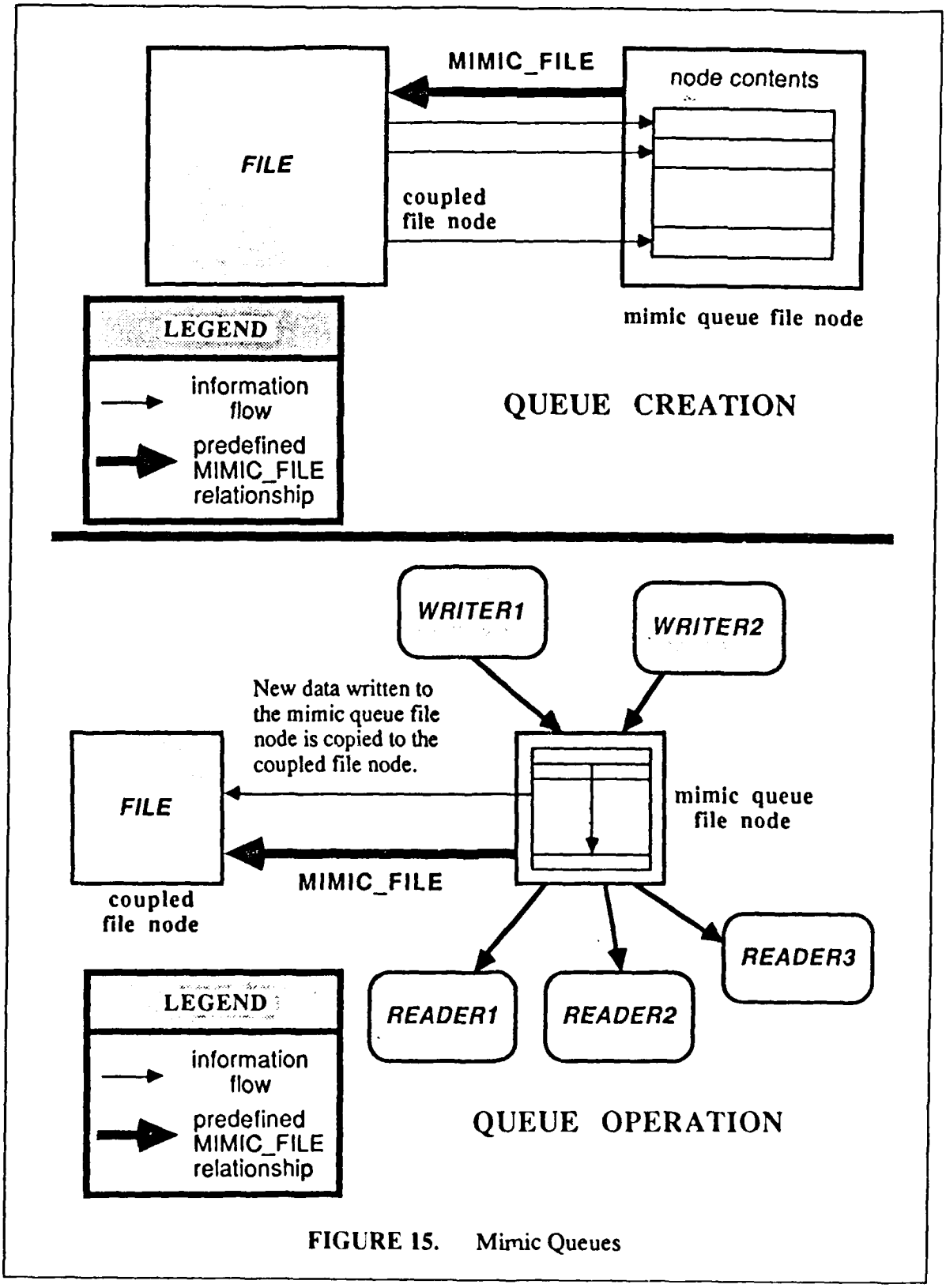


FIGURE 15. Mimic Queues

5.4 Sequential, Direct, and Text Input and Output

Sequential, direct and text input and output are all provided by the Ada language (see Chapter 14 of [1815A]). The CAIS provides much of this same functionality. Interfaces describing operations for direct, sequential and text input and output are discussed in Sections 5.3.4, 5.3.5 and 5.3.6, respectively, of [1838].

6. REFERENCES

[1815A] *Military Standard Ada Programming Language*, United States Department of Defense, ANSI/MIL-STD-1815A, 22 January 1983.

[1838] *Military Standard Common APSE Interface Set (CAIS)*, United States Department of Defense, DOD-STD-1838, 9 October 1986.

[ANSI 73a] American National Standards Institute, *Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI) (ANSI Standard x3.22-1973).*

[ANSI 73b] American National Standards Institute, *Recorded Magnetic Tape for Information Interchange (1600 CPI, PE) (ANSI Standard x3.39-1973).*

[ANSI 76] American National Standards Institute, *Recorded Magnetic Tape for Information Interchange (6250 CPI, Group-coded Recording) (ANSI Standard x3.54-1976).*

[ANSI 78] American National Standards Institute, *Magnetic Tape Labels and File Structure for Information Interchange (ANSI Standard x3.27-1978).*

[APSE] Memorandum from the Office of the Under Secretary of Defense for Research and Engineering, "Ada Programming Support Environment", 15 January 1982.

[CAIS85] *Military Standard Common APSE Interface Set*, United States Department of Defense, Proposed MIL-STD-CAIS, 31 January 1985.

[DCAIS] *Distributing the Common APSE (Ada Programming Support Environment) Interface Set (CAIS)*, MITRE Report MTR-86W00181, The MITRE Corporation, McLean, Virginia, January 1987.

[ISO 76a] *ISO 1863, Information Processing - 9 track, 12.7 mm (0.5 in) wide magnetic tape for information interchange recorded at 32 rpmm (800 cpi).*

[ISO 76b] *ISO 3788, Information Processing - 9 track, 12.7 mm (0.5 in) wide magnetic tape for information interchange recorded at 63 rpmm (1600 cpi) phase encoded.*

[ISO 84] *ISO 5652, Information Processing - 9 track, 12.7 mm (0.5 in) wide magnetic tape for information interchange - Format and recording using group coding at 246 cpmm (6250 cpi).*

[RAC86] *DoD Requirements and Design Criteria for the Common APSE Interface Set (CAIS)*, KIT/KITIA, 4 October 1986.

[RCAIS] *Rehosting the Common APSE (Ada Programming Support Environment) Interface Set (CAIS)*, MITRE Report MTR-86W00198, The MITRE Corporation, McLean, Virginia, January 1987.

[STONEMAN] *Requirements for Ada Programming Support Environments, "STONEMAN"*, February 1980.

[SWAV] "Ada Status and Outlook", *Software for Avionics*, AGARD Conference Proceedings #30, Specialized Printing Services, Looghton, England, January 1983.

[TCSEC] *Department of Defense Trusted Computer System Evaluation Criteria*, Department of Defense Computer Security Center, CSC-STD-001-83, 15 August 1983.

Distribution List for IDA Paper P-2034

NAME AND ADDRESS	NUMBER OF COPIES
Sponsor	
Ms. Virginia Castor Director Ada Joint Program Office (AJPO) 1211 Fern St., Room C-107 Arlington, VA 22202	50 copies
Other	
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2 copies
Ms. Patricia Oberndorf Naval Ocean Systems Center (NOSC) Code 423 San Diego, CA 92152-5000	5 copies

END

DATE

FILMED

MARCH

1988

DTIC